

TABLE1

```

function BloxAPI( context, appName, source, pollInterval, prefixURL )
{
    // Public methods
    this.poll = _poll;
    this.setEnablePolling = _setEnablePolling;
    this.getEnablePolling = _getEnablePolling;
    this.setPollingInterval = _setPollingInterval;
    this.getPollingInterval = _getPollingInterval;
    this.pollTimerHandler = _pollTimerHandler;
    this.sendEvent = _sendEvent;
    this.call = _call;
    this.callBean = _callBean;
    this.addEventListener = _addEventListener;
    this.addErrorHandler = _addErrorHandler;
    this.addBusyHandler = _addBusyHandler;
    this.registerBlox = _registerBlox;
    this.getBlox = _getBlox;
    this.rpc = _rpc;
    this.setDebug = _setDebug;
    this.setBloxBusy = _setBloxBusy;
    this.isSessionExpired = _isSessionExpired;
    this.processSessionExpired = _processSessionExpired;
    this.handleModalDialog = _handleModalDialog;

    // Private data
    this._bloxHttp = new BloxHttp();

    this._pollingEnabled = false;
    this._pollTimeout = null;
    this._eventListeners = new Array();
    this._errorHandlers = new Array();
    this._busyHandlers = new Array();
    this._pendingEvents = new Array();
    this._debug = false;
    this._registeredBlox = new Array();
    this._busyPollIntervalMS = 1000;
    this._serverBusy = true;
    this._initialRequest = true;
    this._sessionExpired = false;
    this._sessionExpiredMessage = "Your session has expired, please refresh the
page";
    this._deferredModalDialogs = new Array();
    this._deferredModelDialogTimer = null;

    // Private methods
    this._startPollTimer = _startPollTimer;
    this._rpc = _rpc;
    this._cancelPollTimer = _cancelPollTimer;
    this._internalRPC = _internalRPC;
    this._processModalDialogs = _processModalDialogs;

    // Begin constructor
    this._context = context;
    this._appName = appName;
    this._pollingIntervalMS = pollInterval;
    this._source = source;
    this._prefixURL = prefixURL;
    this.addErrorHandler( _soapErrorHandler );

    addOnloadMethod( "bloxAPI.setEnablePolling(" + (pollInterval > 0) + ");" );
}

```

```

// End constructor

/**
 * Control debugging mode.
 *
 * @param boolean enable debugging
 */
function _setDebug( debug )
{
    this._debug = debug;
}

/**
 * Add an event listener. Event listeners will receive all events before the
event is sent to the
 * server. The function provided will be passed an event object and should
return true to indicate
 * that the event was handled.
 *
 * @param Function event listener function
 */
function _addEventListener( listener )
{
    this._eventListeners[ this._eventListeners.length ] = listener;
}

/**
 * Add an error handler. Error handlers will receive all soap errors before
they are processed by
 * the client. Use this to override the default error handler behavior. The
function provided will
 * be passed the SimpleSoapResponse object and should return true to indicate
that the error was handled.
 *
 * @param Function error handler function
 */
function _addErrorHandler( handler )
{
    this._errorHandlers[ this._errorHandlers.length ] = handler;
}

/**
 * Add an Blox busy handler. Busy handlers will be invoked whenever a Blox
busy state changes. The
 * JS function provided will be called with the Blox. The busy state on the
Blox indicates it's current
 * busy state. Return true to prevent further processing of the state change.
The default action will
 * grey out the Blox when busy.
 *
 * @param Function busy handler function
 */
function _addBusyHandler( handler )
{
    this._busyHandlers[ this._busyHandlers.length ] = handler;
}

/**
 * Returns true if the Alphablox session has expired.
 *
 * @return boolean true if the session has expired.
 */
function _isSessionExpired( )

```

TABLE1.txt

```

{
    return this._sessionExpired;
}

/**
 * Register a Blox with the API. Registered Blox will receive updates
from the server. If a
 * Blox is not registered, it will not receive any updates from the
server.
 *
 * This is an overloaded method... you can call registerBlox( blox ) and
pass in a JS Blox object,
 * or you can call registerBlox( bloxName, bloxUID ) which will create a new
JS Blox object,
 * save that Blox object on the html element and adds it to the BloxAPI's
blox array.
 */
 * @param Blox blox to be registered
 */
function _registerBlox()
{
    var __blox__;

    // First case creates a new Blox object.
    if (typeof(_registerBlox.arguments[0]) == "string" &&
_registerBlox.arguments.length == 3) {
        var bloxName = _registerBlox.arguments[0];
        var uid = _registerBlox.arguments[1];
        var bloxElement = document.getElementById(uid);

        __blox__ = new Blox( bloxName, uid, _registerBlox.arguments[2]);
        bloxElement.blox = __blox__;
        bloxElement.oncontextmenu = function () { return false; };
    } // Second case assumes that the Blox object is passed in.
    else {
        __blox__ = _registerBlox.arguments[0];
    }

    // Make sure that the Blox has not been previously registered
    for ( var i=0; i < this._registeredBlox.length; i++ )
        if ( __blox__.getName() == this._registeredBlox[ i ].getName() )
            return;

    this._registeredBlox[ this._registeredBlox.length ] = __blox__;

    // Add it to the document object
    eval("document." + __blox__.getName() + " = __blox__;");
    eval("window." + __blox__.getName() + " = __blox__;");
}

/**
 * Lookup blox by name in the registered blox list.
 *
 * @param String blox name
 * @return Blox registered blox or null
 */
function _getBlox( name )
{
    for ( var bloxIdx=0; bloxIdx < this._registeredBlox.length; bloxIdx++ )
        if ( name == this._registeredBlox[bloxIdx].getName() )
            return this._registeredBlox[bloxIdx];
    return null;
}

```

TABLE1.txt

```

}

/**
 *    Connect to a web page and return the result.
 *
 *    @param String url full URL to the web page to connect to
 *    @return String web page text
 */
function _call( url )
{
    // Make sure the browser does not return a cached page
    url += ( url.indexOf( '?' ) == -1 ) ? "?_cb=" : "&_cb=";
    url += new Date().getTime();

    this._bloxHttp.open( "GET", url, false, null, null );
    this._bloxHttp.send( null );
    var response = this._bloxHttp.getResponse();
    this.poll();
    return response;
}

/**
 *    Force a server poll immediately. This will cause any pending client
updates for    registered Blox to be returned.
 *
 */
function _poll( )
{
    // This poll does not use the application context and thus will not keep the
session alive. This is    // desired so that the session can timeout if no real activity occurs.
    this._internalRPC( "AlphabloxServer", new SimpleSoapRequest( "poll"
) );
}

/**
 *    Control the automatic polling of the server.
 *
 *    @param boolean enable automatic server polling|
 */
function _setEnabledPolling( enabled )
{
    this._pollingEnabled = enabled;

    if ( enabled )
        this._startPollTimer( );
    else if ( !enabled )
        this._cancelPollTimer();
}

/**
 *    Return the enabled state of automatic server polling.
 *
 *    @return boolean automatic polling enabled
 */
function _getEnablePolling( )
{
    return this._pollingEnabled;
}

/**
 *    Set the polling interval for non-busy polling. This is the normal
 *    polling "heartbeat" that checks the server for asynchronous updates.

```

```

                                TABLE1.txt
* The polling mechanism uses a different interval when the server
* informs the client that it is busy.
*
* @param int milliseconds number of milliseconds between polls
*/
function _setPollingInterval( milliseconds )
{
    this._pollingIntervalMS = milliseconds;
}

/**
* Return the current non-busy polling interval.
*
* @return int non-bust polling interval
*/
function _getPollingInterval( )
{
    return this._pollingIntervalMS;
}

/**
* Send an event to the server.
*
* @param Event event event to send to the server
*/
function _sendEvent( event )
{
    // Send event to all registered event listeners
    for ( var i=this._eventListeners.length; i-- > 0; )
        if ( this._eventListeners[i]( event ) )
            return false;

    if ( event.isReplaceDuplicate() ) { // Replace all duplicate events by
removing them
        for ( var j=this._pendingEvents.length; j-- > 0; )
            if ( event.isDuplicate( this._pendingEvents[j] ) )
                this._pendingEvents.splice( j, 1 );
        }

    this._pendingEvents[ this._pendingEvents.length ] = event;

    if ( event.isUrgent( ) ) // Send event and keep the session alive
(application poll destination)
        this._rpc( new SimpleSoapRequest( "poll" ) );

    return true;
}

/**
* Call a method on a bean on the server and return the return value of
that method.
*
* @param String beanName name of the bean
* @param String beanMethodName name of the method
* @param Array array of method parameters
* @param Array array of method parameter types
* @return return value from the bean method
*/
function _callBean( beanName, beanMethodName /*, methodParameters,
methodParameterTypes */ )
{
    var soap = new SimpleSoapRequest( "callBean" );
    soap.addParameter( "bean", beanName );
}

```

```

TABLE1.txt
    soap.addParameter( "method", beanMethodName );
    if ( _callBean.arguments.length >= 3 ) {
        var parameters = _callBean.arguments[2];
        var types = ( _callBean.arguments.length == 4 ) ?
_callBean.arguments[3] : null;
        for ( var i=0; i < parameters.length; i++ ) {
            var name = "arg" + (i+1);
            if ( types != null && i < types.length )
                soap.addParameter( name, parameters[i], types[i] );
            else
                soap.addParameter( name, parameters[i] );
        }
        return this._rpc( soap );
    }
    /**
    * Send a SOAP RPC request to the server and process the results. This
method will
    * invoke method indicated in the RPC package and return the method
result.
    *
    * @param SoapRequest soapRequest the request to be sent to the server
    * @return return value from the invoked method
    */
    function _rpc( soapRequest )
    {
        return this._internalRPC( this._context + "/abx", soapRequest );
    }

    /**
    * Called by a Blox to visually indicated that it is busy.
    *
    * @param Blox
    * @return boolean busy handled (no default behavior)
    */
    function _setBloxBusy( blox )
    {
        // Send event to all registered event listeners
        for ( var i=this._busyHandlers.length; i-- > 0; )
            if ( this._busyHandlers[i]( blox ) )
                return true;

        return false;
    }

    /**
    * Call this method to force a session expired state for all Blox managed by
this
    * BloxAPI. One this method is called, the Blox will no longer generate RPC
calls
    * and will be disabled.
    */
    function _processSessionExpired( )
    {
        this._sessionExpired = true;
        for ( var bloxIdx=0; bloxIdx < this._registeredBlox.length;
bloxIdx++ ) {

```

```

TABLE1.txt
var blox = this._registeredBlox[bloxIdx];

blox.getContainer().title = this._sessionExpiredMessage;
    blox.closeAllDialogs( );
blox.setDisabled( true );
blox.setBusyLogo( false );
}
}

////////////////////////////////////
////////////////////////////////////
// Private Method Definitions

// Invoke a server RPC
function _internalRPC( URLPath, soapRequest )
{
    if ( this._sessionExpired )                // If the session
has expired we are done
        return null;

    this._cancelPollTimer( );

    soapRequest.setSource( this._source );
    soapRequest.setContext( this._appName);
    soapRequest.setInitialRequest( this._initialRequest );
    soapRequest.setThemeName( _themeName );

    this._initialRequest = false;

    // Add all pending events to the request
    for ( var e=0; e < this._pendingEvents.length; e++ )
        soapRequest.addEvent( this._pendingEvents[e] );
    this._pendingEvents.length = 0;

    // Add an update header for each blox managed by this Blox API
    for ( var bloxIdx=0; bloxIdx < this._registeredBlox.length;
bloxIdx++ )
        soapRequest.addUpdateHeader( this._registeredBlox[bloxIdx].getName()
);

    if ( this._debug )
        alert( "SOAP Request: " + _prefixURL + "/" + URLPath +
"/soap/" + "\r\n\r\n" + soapRequest.getRequest() );

    var soapResponse = null;

    try
    {
        bloxHttp.open( "POST", _prefixURL + "/" + URLPath + "/soap/", false,
null, null );
        bloxHttp.send( soapRequest.getRequest() );

        var response = bloxHttp.getResponse();
        var status = bloxHttp.getStatus( );
        var document = bloxHttp.getResponseXML( ).documentElement;

        if ( status != 200 ) {                // We never seem to get here as the browser
catches the error, but just in case ...
            alert( "An unrecoverable error occured communicating with the
Alphablox server" );
            return null;
        }
    }
}

```

TABLE1.txt

```

    if ( this._debug )
        alert( "SOAP Response (status " + status + "):\r\n\r\n" + response
);

    // Process response headers (blox busy, dialog changes, model changes
    if ( document != null ) {
        soapResponse = new SimpleSoapResponse( document );
    }
    else {
        soapResponse = new SimpleSoapResponse( );
        soapResponse.faultCode = "env:Receiver";
        soapResponse.faultReason = "Unable to parse the server's response
(document is null)\r\n\r\n" + response;
    }
    catch ( e )
    {
        var problem = e.description;
        if ( problem == null || problem.length == 0 )
            problem = "Communications error";
        else
            problem += " [" + e.name + "]";

        soapResponse = new SimpleSoapResponse( );
        soapResponse.faultCode = "env:Receiver";
        soapResponse.faultReason = "Trouble communicating with the Alphablox
Server\r\n\r\nCause: " + problem;
    }

    if ( soapResponse.hasFault() ) {
        // If the session has ended, close all dialogs and disable all
        if ( "blox:sessionexpired" == soapResponse.faultSubcode )
            this.processSessionExpired( );

        // Send error to all registered error handlers till it is handled
        for ( var j=this._errorHandlers.length; j-- > 0 ; )
            if ( this._errorHandlers[j]( soapResponse ) )
                break;

        if ( "blox:sessionexpired" != soapResponse.faultSubcode ) {
            this._serverBusy = false;
            this._startPollTimer( );
        }

        return null;
    }

    // Refresh the client state if needed
    if ( soapResponse.isRefreshClient( ) )
        _refreshClientState( );

    // Deal with new browser windows
    soapResponse.dispatchBrowserWindows( );

    var serverBusy = false;

    for ( var i=0; i < this._registeredBlox.length; i++ ) {
        var blox = this._registeredBlox[i];

        // Dispatch all request headers for this Blox

```



```

                                TABLE1.txt
                                soapResponse.dispatchClosedDialogs( blox );
                                soapResponse.dispatchComponentUpdates( blox );
                                soapResponse.dispatchDialogComponentUpdates( blox );
                                soapResponse.dispatchRightClickMenus( blox );
                                soapResponse.dispatchBloxBusyStatus( blox );

                                // Deal with the blox busy state
                                serverBusy = serverBusy || blox.isBusy( );
                                }

                                soapResponse.dispatchClipboardContents( );
                                soapResponse.dispatchClientCommands( );

                                this._serverBusy = serverBusy;
                                this._startPollTimer( );

                                // Must be last due to modal dialog behavior
                                for ( var d=0; d < this._registeredBlox.length; d++ )
                                    soapResponse.dispatchNewDialogs( this._registeredBlox[d] );

                                return soapResponse.getResponse();
                                }

                                // Cancel the poll timer
                                function _cancelPollTimer( )
                                {
                                    if ( this._pollTimeout != null ) {
                                        clearTimeout( this._pollTimeout );
                                        this._pollTimeout = null;
                                    }
                                }

                                // Handle an automatic poll
                                function _pollTimerHandler()
                                {
                                    _cancelPollTimer( );

                                    if ( !this._pollingEnabled )
                                        return;

                                    this.poll( );
                                }

                                // Start the automatic poll timer
                                function _startPollTimer( )
                                {
                                    if ( this._pollingEnabled && this._pollTimeout == null ) {
                                        var interval;

                                        if ( this._initialRequest )                // Send the first
poll out immediately to collect any dialogs
                                            interval = 0;
                                        else
                                            interval = ( this._serverBusy ) ?
this._busyPollIntervalMS : this._pollingIntervalMS;

                                        this._pollTimeout = setTimeout(
"bloxAPI.pollTimerHandler()", interval );
                                    }
                                }

                                // Default error handler
                                function _soapErrorHandler( soapResponse )

```

TABLE1.txt

```

{
    alert( "Alphablox Server:\r\n\r\n" + soapResponse.faultReason );
    return true;
}

// Handle modal dialogs
function _handleModalDialog( dialog )
{
    if ( isModalDialogDisplayed() ) {
        dialog.open();
        return;
    }

    this._deferredModalDialogs.push( dialog );

    if ( this._deferredModelDialogTimer == null )
        this._deferredModelDialogTimer = setInterval(
"bloxAPI._processModalDialogs( );", 20 );
}

function _processModalDialogs( )
{
    if ( this._deferredModalDialogs.length == 0 ) {
        if ( this._deferredModelDialogTimer != null )
            clearInterval( this._deferredModelDialogTimer );
        this._deferredModelDialogTimer = null;
        return;
    }

    // If here are any onload methods then wait
    if ( onloadMethods.length > 0 )
        return;

    // If there are any deffered component updates then wait
    for ( var i=0; i < this._registeredBlox.length; i++ ) {
        var blox = this._registeredBlox[i];
        if ( blox._deferredUpdates.length > 0 )
            return;
    }

    clearInterval( this._deferredModelDialogTimer );
    this._deferredModelDialogTimer = null;

    // Process all modal dialogs
    while ( this._deferredModalDialogs.length > 0 ) {
        this._deferredModalDialogs.pop().open();
    }
}

if ( !bloxAPI )
{
    var bloxAPI = new BloxAPI( _appContext, _appName, _renderType, _pollInterval ,
_prefixURL );
}

/**
 *      Exception class
 */
function Exception( exceptionClass, message )
{
    // Public methods
    this.getExceptionClass = _getExceptionClass;
}

```

```
this.getMessage = _getMessage;
this.toString = _toString;

    // Private data

// Private methods

// Begin constructor
    this._exceptionClass = exceptionClass;
this._message = message;
// End constructor

function _getExceptionClass( )
{
    return this._exceptionClass;
}

function _getMessage( )
{
    return this._message;
}

function _toString( )
{
    return this._exceptionClass + ": " + this._message;
}
}
```

TABLE 2

```

/**
 *      BLOX HTTP SOAP request "class".
 */
function SimpleSoapRequest( /* String */ methodName )
{
    // Public methods
    this.getRequest = _getRequest;
    this.addParameter = _addParameter;
    this.setBloxName = _setBloxName;
    this.getBloxName = _getBloxName;
    this.setSource = _setSource;
    this.addUpdateHeader = _addUpdateHeader;
    this.setContext = _setContext;
    this.setInitialRequest = _setInitialRequest;
    this.setThemeName = _setThemeName;
    this.addEvent = _addEvent;

    // Private data
    this._parameters = new Array();
    this._bloxName = null;
    this._source = null;
    this._context = null;
    this._updateList = null;;
    this._initialRequest = false;
    this._themeName = null;
    this._events = null;
    // Private methods

    // Begin constructor
    this.methodName = methodName;
    // End constructor

    function _getRequest( )
    {
        var request = new Array();
        request.push( "<?xml version=\"1.0\" encoding=\"UTF-8\" ?>\r\n" );
        request.push( "<env:Envelope"
xmlns:env=\"http://www.w3.org/2002/06/soap-envelope\"
xmlns:blox=\"http://alphablox.com/soap-request\"/>\r\n" );
        request.push( "
xmlns:xsd=\"http://www.w3.org/2001/09/soap-encoding\">\r\n" );
        request.push( "    <env:Header>\r\n" );
        request.push( "        <blox:routing>\r\n" );
        request.push( "            <blox:handler>RenderManager</blox:handler>\r\n" );

        if ( this._source != null ) {
            request.push( "                <blox:source>" );
            request.push( this._source );
            request.push( "</blox:source>\r\n" );
        }

        if ( this._context != null ) {
            request.push( "                <blox:context>" );
            request.push( this._context );
            request.push( "</blox:context>\r\n" );
        }

        if ( this._bloxName != null ) {
            request.push( "                <blox:bloxname>" );
            request.push( this._bloxName );
            request.push( "</blox:bloxname>\r\n" );
        }
    }
}

```

TABLE2.txt

```

if ( this._themeName != null ) {
    request.push( "        <blox:theme>" );
    request.push( this._themeName );
    request.push( "</blox:theme>\r\n" );
}

if ( this._initialRequest )
    request.push( "        <blox:initialRequest />\r\n" );

request.push( "        </blox:routing>\r\n" );

if ( this._updateList != null && this._updateList.length > 0 ) {
    request.push( "        <blox:update>\r\n" );

    for ( var i=0; i < this._updateList.length; i++ ) {
        request.push( "            <blox:bloxname>" );
        request.push( this._updateList[i] );
        request.push( "</blox:bloxname>\r\n" );
    }

    request.push( "        </blox:update>\r\n" );
}

if ( this._events != null && this._events.length > 0 ) {
    for ( var e=0; e < this._events.length; e++ ) {
        var event = this._events[e];

        request.push( "            <blox:event>\r\n" );
        request.push( "                <blox:class>" + event.getEventClass() +
"</blox:class>\r\n" );
        request.push( "                <blox:bloxname>" + event.getBloxName() +
"</blox:bloxname>\r\n" );

        if ( event.getDestinationUID() != null && event.getDestinationUID()
> 0 ) // If there is a UID, use it
            request.push( "                    <blox:uid>" + event.getDestinationUID()
+ "</blox:uid>\r\n" );
        else if ( event.getDestinationName() != null ) // If there is
no UID but there is a name, use the name
            request.push( "                    <blox:name>" +
event.getDestinationName() + "</blox:name>\r\n" );

        // Add any event-specific attributes
        var attrs = event.getAllAttributes();
        for ( var e1=0; e1 < attrs.length; e1++ ) {
            request.push( '                <blox:param name="' + attrs[e1][0] +
'">\r\n' );
            request.push( "                    " + _dataToXML( "value",
attrs[e1][1], null ) + "\r\n" );
            request.push( "                </blox:param>\r\n" );
        }

        request.push( "            </blox:event>\r\n" );
    }
}

request.push( "    </env:Header>\r\n    <env:Body>\r\n" );
request.push( "        <blox:" + methodName + ">\r\n" );

for ( var j=0; j < this._parameters.length; j++ ) {
    var parameter = this._parameters[j];
    request.push( "            " );
}

```

```

                                TABLE2.txt
parameter[2] ) );      request.push( _dataToXML( parameter[0], parameter[1],
                        request.push( "\r\n" );
                        }

                        request.push( "        </blox:" );
                        request.push( methodName );
                        request.push( ">\r\n" );
                        request.push( "    </env:Body>\r\n</env:Envelope>\r\n" );

                        return request.join( "" );
                    }

function _dataToXML( name, data, type )
{
    var request = new Array( );

    if ( data == null ) {
        request.push( "<blox:" + name + " xsd:null=\"true\"/>" );
    }
    else if ( typeof data != "object" ) {
        var cdata = true;
        request.push( "<blox:" + name );
        if ( type != null ) {
            request.push( " xsd:type=\"" );
            request.push( type );
            request.push( "\" );

            if ( type == "boolean" || type == "int" || type == "integer"
|| type == "long" || type == "byte" ||
                type == "double" || type == "float" )
                cdata = false;
        }
        request.push( ">" );

        if ( cdata ) request.push( "<![CDATA[" );
        request.push( data );
        if ( cdata ) request.push( "]">" );

        request.push( "</blox:" );
        request.push( name );
        request.push( ">" );
    }
    else {
        // Handle arrays
        request.push( "<blox:" );
        request.push( name );
        request.push( " xsd:type=\"Array\">" );

        for ( var j=0; j < data.length; j++ )
            request.push( _dataToXML( name + j, data[j], type )

);

        request.push( "</blox:" );
        request.push( name );
        request.push( ">" );
    }

    return request.join( "" );
}

function _addParameter( name, value /*, type */ )
{
    var index = this._parameters.length;

```

```

                                TABLE2.txt
    var type = null;
    if ( _addParameter.arguments.length >= 3 )
        type = _addParameter.arguments[2];
    this._parameters[ index ] = new Array( name, value, type );
}
function _addEvent( event )
{
    if ( this._events == null )
        this._events = new Array();
    this._events[ this._events.length ] = event;
}

function _setBloxName( name )
{
    this._bloxName = name;
}

function _getBloxName( )
{
    return this._bloxName;
}

function _setSource( source )
{
    this._source = source;
}

function _setContext( context )
{
    this._context = context;
}

function _setInitialRequest( initialRequest )
{
    this._initialRequest = initialRequest;
}

function _setThemeName( themeName )
{
    this._themeName = themeName;
}

function _addUpdateHeader( bloxName )
{
    if ( bloxName == null || typeof bloxName != "string" ||
bloxName.length == 0 )
        return;

    if ( this._updateList == null )
        this._updateList = new Array();

    var index = this._updateList.length;
    this._updateList[ index ] = bloxName;
}

}

/**
 * BLOX HTTP SOAP response "class".
 */

```

TABLE2.txt

```

function SimpleSoapResponse( /* DOMDocument */ responseDoc )
{
    // Public methods
    this.getResponse = _getResponse;
    this.dispatchBloxBusyStatus = _dispatchBloxBusyStatus;
    this.dispatchClosedDialogs = _dispatchClosedDialogs;
    this.dispatchComponentUpdates = _dispatchComponentUpdates;
    this.dispatchDialogComponentUpdates = _dispatchDialogComponentUpdates;
    this.dispatchNewDialogs = _dispatchNewDialogs;
    this.dispatchRightClickMenus = _dispatchRightClickMenus;
    this.dispatchBrowserWindows = _dispatchBrowserWindows;
    this.dispatchClientCommands = _dispatchClientCommands;
    this.dispatchClipboardContents = _dispatchClipboardContents;
    this.isRefreshClient = _isRefreshClient;
    this.hasFault = _hasFault;

    // Public properties
    this.faultReason = null;
    this.faultCode = null;
    this.faultSubcode = null;

    // Private data
    this._busyHeaders = null;
    this._componentUpdates = null;
    this._dialogUpdates = null;
    this._newDialogs = null;
    this._closedDialogs = null;
    this._rightClickMenu = null;
    this._browserWindows = null;
    this._clientCommands = null;
    this._clipboardContents = null;
    this._result = null;
    this._refreshClient = false;

    // Private methods

    // Begin constructor
    if ( responseDoc == null )
        return;

    var headerNodes = responseDoc.getElementsByTagName( "env:Header" );
    if ( headerNodes != null && headerNodes.length == 1 ) {
        var header = headerNodes.item( 0 );

        this._busyHeaders = _parseBusyHeaders( header.getElementsByTagName(
"blox:busyState" ) );
        this._componentUpdates = _parseComponentUpdateHeaders(
header.getElementsByTagName( "blox:componentUpdate" ) );
        this._dialogUpdates = _parseDialogUpdateHeaders(
header.getElementsByTagName( "blox:updateDialog" ) );
        this._closedDialogs = _parseClosedDialogHeaders(
header.getElementsByTagName( "blox:closeDialog" ) );
        this._newDialogs = _parseNewDialogHeaders(
header.getElementsByTagName( "blox:newDialog" ) );
        this._rightClickMenu = _parseRightClickMenuHeaders(
header.getElementsByTagName( "blox:rightClickMenu" ) );
        this._browserWindows = _parseBrowserWindowHeaders(
header.getElementsByTagName( "blox:browserwindow" ) );
        this._clientCommands = _parseClientCommandHeaders(
header.getElementsByTagName( "blox:clientCommand" ) );
        this._clipboardContents = _parseClipboardHeaders(
header.getElementsByTagName( "blox:clipboardContents" ) );
    }
}

```


TABLE2.txt

```

var refreshNodes = responseDoc.getElementsByTagName( "blox:refreshClient" );
if ( refreshNodes != null && refreshNodes.length == 1 )
    this._refreshClient = true;
}

var bodyNodes = responseDoc.getElementsByTagName( "env:Body" );
if ( bodyNodes != null && bodyNodes.length == 1 ) {
    var body = bodyNodes.item( 0 );

    // Check for a fault in the body
    var faultNodes = body.getElementsByTagName( "env:Fault" );
    if ( faultNodes != null && faultNodes.length > 0 ) {
        this.faultCode = "";

        var codeNodes = faultNodes.item(0).getElementsByTagName(
"env:Code" );
        if ( codeNodes != null && codeNodes.length > 0 ) {
            var n1 = codeNodes.item(0).getElementsByTagName(
"env:Value" );
            if ( n1 != null && n1.length > 0 )
                this.faultCode = n1.item(0).text;
        }

        var subcodeNodes = faultNodes.item(0).getElementsByTagName(
"env:Subcode" );
        if ( subcodeNodes != null && subcodeNodes.length > 0 ) {
            var n2 = subcodeNodes.item(0).getElementsByTagName(
"env:Value" );
            if ( n2 != null && n2.length > 0 )
                this.faultSubcode = n2.item(0).text;
        }

        var reasonNodes = faultNodes.item(0).getElementsByTagName(
"env:Reason" );
        if ( reasonNodes != null && reasonNodes.length > 0 )
            this.faultReason = reasonNodes.item(0).text;
        else {
            var child = body.firstChild;
            if ( child != null ) {
                var resultNodes = child.getElementsByTagName(
"blox:result" );
                if ( resultNodes != null && resultNodes.length == 1 )
                    this._result = _parseResultData( resultNodes.item( 0 ) );
            }
        }
    }
}

// End constructor

function _hasFault( )
{
    return this.faultCode != null;
}

function _isRefreshClient( )
{
    return this._refreshClient;
}

function _dispatchBloxBusystatus( blox )

```

TABLE2.txt

```

{
    var bloxName = blox.getName();
    if ( this._busyHeaders != null ) {
        for ( var i=0; i < this._busyHeaders.length; i++ ) {
            var header = this._busyHeaders[i];
            if ( bloxName == header[0] ) {
                setTimeout( bloxName + ".handleBusy( " +
header[1] + " );", 0 );
                return;
            }
        }
    }
}

function _dispatchComponentUpdates( blox )
{
    var bloxName = blox.getName();
    if ( this._componentUpdates != null ) {
        var updates = false;
        for ( var i=0; i < this._componentUpdates.length; i++ ) {
            var header = this._componentUpdates[i];
            if ( bloxName == header[0] ) {
                blox.handleComponentUpdate( header[1], header[2], header[3] );
                updates = true;
            }
        }
        if ( updates ) {
            blox.endComponentUpdates( );
        }
    }
}

function _dispatchDialogComponentUpdates( blox )
{
    var bloxName = blox.getName();
    if ( this._dialogUpdates != null ) {
        var updates = false;
        for ( var i=0; i < this._dialogUpdates.length; i++ ) {
            var header = this._dialogUpdates[i];
            if ( bloxName == header[0] ) {
                blox.handleDialogComponentUpdate( header[1],
header[2], header[3] );
                updates = true;
            }
        }
        if ( updates ) {
            blox.endDialogComponentUpdates( );
        }
    }
}

function _dispatchClosedDialogs( blox )
{
    var bloxName = blox.getName();
    if ( this._closedDialogs != null ) {

```

```

TABLE2.txt
    for ( var i=0; i < this._closedDialogs.length; i++ ) {
        var header = this._closedDialogs[i];
        if ( bloxName == header[0] )
            blox.handleCloseDialog( header[1] );
    }
}

function _dispatchNewDialogs( blox )
{
    var bloxName = blox.getName();

    if ( this._newDialogs != null ) {
        for ( var i=0; i < this._newDialogs.length; i++ ) {
            var header = this._newDialogs[i];
            if ( bloxName == header[0] )
                blox.handleNewDialog( header[1], header[2],
header[3], header[4], header[5], header[6], header[7] );
        }
    }

    function _dispatchRightClickMenus( blox )
    {
        var bloxName = blox.getName();

        if ( this._rightClickMenu != null ) {
            for ( var i=0; i < this._rightClickMenu.length; i++ ) {
                var header = this._rightClickMenu[i];
                if ( bloxName == header[0] )
                    blox.handleRightClickMenu( header[1],
header[2], header[3], header[4] );
            }
        }

        function _dispatchBrowserWindows( )
        {
            if ( this._browserwindows != null )
                for ( var i=0; i < this._browserwindows.length; i++ ) {
                    var browser = this._browserwindows[i];
                    var win;
                    if ( browser[2] == null )
                        win = window.open( browser[0], browser[1] );
                    else
                        win = window.open( browser[0], browser[1], browser[2] );

                    try {
                        if ( win != null )
                            win.focus();
                    } catch ( e ) {
                        // when setting focus to an existing window, we will get a
bizzare exception
                    }
                }
        }

        function _dispatchClientCommands( )
        {
            if ( this._clientCommands != null ) {
                for ( var i=0; i < this._clientCommands.length; i++ ) {
                    var command = this._clientCommands[i];
                    _soapClientCommands[ _soapClientCommands.length ] = command[0];
                }
            }
        }
    }
}

```

```

    }

    if ( _soapClientCommands.length > 0 && _soapClientCommandTimer == null )
        _soapClientCommandTimer = setTimeout(
            "_runNextSoapClientCommand();", 0 );
    }

    function _dispatchClipboardContents( )
    {
        if ( this._clipboardContents != null ) {
            for ( var i=0; i < this._clipboardContents.length; i++ ) {
                var contents = this._clipboardContents[i];
                window.clipboardData.setData( contents[0], contents[1] );
            }
        }
    }

    function _getResponse( )
    {
        return this._result;
    }

    function _parseBusyHeaders( busyNodes )
    {
        var headers = new Array();
        for ( var i=0; i < busyNodes.length; i++ ) {
            var busy = busyNodes.item( i );

            headers[ headers.length ] =
                new Array( busy.getAttribute( "blox:bloxName" ),
                    busy.getAttribute( "blox:busy" ) ==
"true" );
        }
        return headers;
    }

    function _parseComponentUpdateHeaders( updateNodes )
    {
        var headers = new Array();
        for ( var i=0; i < updateNodes.length; i++ ) {
            var update = updateNodes.item( i );

            var contentNodes = update.getElementsByTagName(
"blox:contents" );
            var contents = "";
            if ( contentNodes != null && contentNodes.length == 1 )
                contents = contentNodes.item(0).text;

            headers[ headers.length ] =
                new Array( update.getAttribute( "blox:bloxName" ),
                    parseInt( update.getAttribute(
"blox:uid" ) ),
                    contents,
                    update.getAttribute( "blox:defer" )
== "true" );
        }
        return headers;
    }

    function _parseNewDialogHeaders( updateNodes )
    {
        var headers = new Array();

```

```

TABLE2.txt
for ( var i=0; i < updateNodes.length; i++ ) {
    var update = updateNodes.item( i );

    var titleNodes = update.getElementsByTagName( "blox:title"
);
    var title = "";
    if ( titleNodes != null && titleNodes.length == 1 )
        title = titleNodes.item(0).text;

    var contentNodes = update.getElementsByTagName(
"blox:contents" );
    var contents = "";
    if ( contentNodes != null && contentNodes.length == 1 )
        contents = contentNodes.item(0).text;

    headers[ headers.length ] =
        new Array( update.getAttribute( "blox:bloxName" ),
                    parseInt( update.getAttribute(
"blox:dialoguid" ) ),
                    update.getAttribute( "blox:modal" )
                    parseInt( update.getAttribute(
"blox:height" ) ),
                    parseInt( update.getAttribute(
"blox:width" ) ),
                    title,
                    contents,
                    update.getAttribute(
"blox:resizeable" ) == "true" );
    }
    return headers;
}

function _parseDialogUpdateHeaders( updateNodes )
{
    var headers = new Array();
    for ( var i=0; i < updateNodes.length; i++ ) {
        var update = updateNodes.item( i );

        var contentNodes = update.getElementsByTagName(
"blox:contents" );
        var contents = "";
        if ( contentNodes != null && contentNodes.length == 1 )
            contents = contentNodes.item(0).text;

        headers[ headers.length ] =
            new Array( update.getAttribute( "blox:bloxName" ),
                        parseInt( update.getAttribute(
"blox:dialoguid" ) ),
                        update.getAttribute( "blox:modal" )
                        parseInt( update.getAttribute(
"blox:height" ) ),
                        parseInt( update.getAttribute(
"blox:width" ) ),
                        contents );
    }
    return headers;
}

function _parseClosedDialogHeaders( updateNodes )
{
    var headers = new Array();
    for ( var i=0; i < updateNodes.length; i++ ) {
        var update = updateNodes.item( i );

        headers[ headers.length ] =

```

```

TABLE2.txt
new Array( update.getAttribute( "blox:bloxName" ),
           parseInt( update.getAttribute(
"blox:dialoguid" ) ) );
    }
    return headers;
}

function _parseRightClickMenuHeaders( updateNodes )
{
    var headers = new Array();
    for ( var i=0; i < updateNodes.length; i++ ) {
        var update = updateNodes.item( i );

        var contentNodes = update.getElementsByTagName(
"blox:contents" );
        var contents = "";
        if ( contentNodes != null && contentNodes.length == 1 )
            contents = contentNodes.item(0).text;

        headers[ headers.length ] =
            new Array( update.getAttribute( "blox:bloxName" ),
                      parseInt( update.getAttribute(
"blox:uid" ) ),
                      parseInt( update.getAttribute(
"blox:xpos" ) ),
                      parseInt( update.getAttribute(
"blox:ypos" ) ),
                      contents );
    }
    return headers;
}

function _parseBrowserWindowHeaders( updateNodes )
{
    var headers = new Array();
    for ( var i=0; i < updateNodes.length; i++ ) {
        var update = updateNodes.item( i );

        var target = update.getAttribute( "blox:windowTarget" );
        var url = "";
        var features = null;

        var urlNodes = update.getElementsByTagName( "blox:windowURL"
);
        if ( urlNodes != null && urlNodes.length == 1 )
            url = urlNodes.item(0).text;

        var featureNodes = update.getElementsByTagName(
"blox:windowFeatures" );
        if ( featureNodes != null && featureNodes.length == 1 )
            features = featureNodes.item(0).text;

        headers[ headers.length ] = new Array( url, target, features
);
    }
    return headers;
}

function _parseClientCommandHeaders( updateNodes )
{
    var headers = new Array();
    for ( var i=0; i < updateNodes.length; i++ ) {

```

TABLE2.txt

```

var command = null;

"blox:command" );
    var commandNodes = update.getElementsByTagName(
        if ( commandNodes != null && commandNodes.length == 1 )
            command = commandNodes.item(0).text;

        headers[ headers.length ] = new Array( command );
    }
    return headers;
}

function _parseClipboardHeaders( clipboardNodes )
{
    var headers = new Array();
    for ( var i=0; i < clipboardNodes.length; i++ ) {
        var clipboard = clipboardNodes.item( i );
        var format = clipboard.getAttribute( "blox:format" );
        var contents = null;

        var contentNodes = clipboard.getElementsByTagName(
            "blox:contents" );
            if ( contentNodes != null && contentNodes.length == 1 )
                contents = contentNodes.item(0).text;

            headers[ headers.length ] = new Array( format, contents );
        }
        return headers;
    }

function _parseResultData( resultNode )
{
    var type = resultNode.getAttribute( "xsd:type" );
    if ( type == null )
        type = resultNode.getAttribute( "blox:type" );
    if ( type == null )
        type = "string";

    if ( type == "string" || type == "Object" )
        return resultNode.text;

    if ( type == "boolean" )
        return "true" == resultNode.text;

    if ( type == "int" || type == "integer" || type == "long" )
        return parseInt( resultNode.text );

    if ( type == "byte" )
        return resultNode.item( 0 ).text.charAt( 0 );

    if ( type == "double" || type == "float" )
        return parseFloat( resultNode.text );

    if ( type == "exception" ) {
        var exception = "";
        var reason = "";

        var nodes = resultNode.getElementsByTagName(
            "blox:exception" );
            if ( nodes != null && nodes.length == 1 ) {
                exception = nodes.item(0).getAttribute( "blox:class"
);

```

```

                                TABLE2.txt
                                reason = nodes.item(0).text;
                                }

                                return new Exception( exception, reason );
                                }

                                if ( type == "Array" ) {
                                    var array = new Array( );
                                    for ( var i=0; i < resultNode.childNodes.length; i++ )
                                        array[i] = _parseResultData( resultNode.childNodes.item( i ) );
                                    return array;
                                }

                                return null;
                            }
                        }

var _soapClientCommands = new Array();
var _soapClientCommandTimer = null;

function _runNextSoapClientCommand( )
{
    _soapClientCommandTimer = null;

    if ( _soapClientCommands.length > 0 ) {
        var command = _soapClientCommands.shift();
        eval( command );
    }

    if ( _soapClientCommands.length > 0 ) {
        _soapClientCommandTimer = setTimeout( "_runNextSoapClientCommand();", 0 );
    }
}

```